НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

_____ Сергій СТІРЕНКО

«___» _____ 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Система управління збору даних у хмарному середовищі»

Виконав:
студент IV курсу, групи ІП-62
Величко Антон Андрійович _____

Керівник:
професор, доктор технічних наук
Сімоненнко Валерій Павлович _____

Консультант з нормокнтролю:
професор, доктор технічних наук
Сімоненнко Валерій Павлович _____

Рецензент:
_____
_____ _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___» _____2020 р.

ЗАВДАННЯ
на дипломний проєкт студенту
Величку Антону Андрійовичу

1. Тема проєкту «Система управління збору даних у хмарному середовищі», керівник проєкту Сімоненнко Валерій Павлович, доктор технічних наук, професор, затверджені наказом по університету від «07» травня 2020 р. № 1081-с

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту: теоретичні відомості, технічна документація, система управління збору даних у хмарному середовищі.

4. Зміст пояснювальної записки: огляд існуючих систем управління збору даних у хмарному середовищі, проєктування системи управління збору даних у хмарному середовищі, реалізація системи управління збору даних у хмарному середовищі, огляд створеної системи управління збору даних у хмарному середовищі.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо): структурна схема системи, схема алгоритму, діаграма класів.

6. Консультанти розділів проекту

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормконтроль	*Сімоненко Валерій Павлович, професор ОТ, д.т.н.*		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Затвердження теми роботи	16.12.2020 - 20.12.2020	
2	Вивчення та аналіз завдання	20.12.2020 - 03.02.2020	
3	Розробка архітектури та загальної структури системи	03.02.2020 - 24.02.2020	
4	Розробка структур окремих підсистем	24.02.2020 - 16.03.2020	
5	Програмна реалізація системи	16.03.2020 - 11.05.2020	
6	Виправлення помилок	11.05.2020 - 01.06.2020	
7	Оформлення пояснювальної записки	03.06.2020	
8	Передзахист	12.06.2020	
9	Захист	15.06.2020 - 19.06.2020	

Студент _____ _____Величко А.А._____
(підпис) (прізвище та ініціали)

Керівник проекту _____ _____Сімоненко В.П._____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота виконана на 51 сторінці і містить 9 ілюстрацій, 9 таблиці, 3 додатки. При розробці використано інформацію з 15 джерел.

Метою даної роботи є створення автоматизованої системи управління збором даних в хмарному середовищі, яка б враховувала всі необхідні параметри в даного процесу.

В дипломній роботі розроблено автоматизовану систему моніторингу, що має надавати зручний функціонал для відслідковування основних процесів системи.

Планування та розподіл ресурсів необхідних для повноцінного процесу збору даних залежить не тільки від кількості даних, що необхідно зібрати, а й від обмежень веб-ресурсів та мережі Інтернет загало.

У розділі програмного забезпечення описані основні засоби розробки комплексу завдань і визначені вимоги до технічного забезпечення, обґрунтована архітектура програмного забезпечення.

Ключові слова: система управління, збір даних, хмарне середовище, моніторинг, планування зборів

АННОТАЦИЯ

Дипломная работа выполнена на 51 странице и содержит 9 иллюстраций, 9 таблицы, 3 приложений. При разработке использована информация из 15 источников.

Целью данной работы является создание автоматизированной системы управления сбором данных в облачной среде, которая бы учитывала все необходимые параметры данного процесса.

В дипломной работе разработана автоматизированная система мониторинга, которая должна предоставлять собой удобный функционал для отслеживания основных процессов системы.

Планирование и распределение ресурсов, необходимых для полноценного процесса сбора данных зависит не только от количества данных, что необходимо собрать, но и от ограничений веб-ресурсов и сети Интернет в целом.

В разделе программного обеспечения описаны основные средства разработки комплекса задач и определены требования к техническому обеспечению, обоснованная архитектура программного обеспечения.

Ключевые слова: система управления, сбор данных, облачная среда, мониторинг, планирование соборов

ANNOTATION

The research paper performed at 51 pages and contains 9 illustrations, 9 tables, 3 appendices. In developing uses information from 15 sources.

The purpose of this work is to create an automated cloud-based data scraping management system, which would take into account all the necessary parameters in this process.

In the thesis an automated monitoring system is developed, which should provide convenient functionality for monitoring the main processes of the system.

The planning and allocation of resources required for a complete data collection process depends not only on the amount of data that needs to be collected, but also on the limitations of web resources and the Internet in general.

The software section describes the main tools for developing a set of tasks and defines the requirements for hardware, substantiates the software architecture.

Keywords: management system, data scpraping, cloud environment, monitoring, collection scheduling

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість	№ докум.	Примітки
		ІАЛЦ.467200.002 ТЗ	Система управління збору	3		
			даних у хмарному			
			середовищі			
			Технічне завдання.			
		ІАЛЦ.467200.003 ВП	Система управління збору	1		
			даних у хмарному			
			середовищі			
			Відомість проекту.			
	А4	ІАЛЦ.467200.004 ПЗ	Система управління збору	51		
			даних у хмарному			
			середовищі			
			Пояснювальна записка			
	А1	ІАЛЦ.467200.005 Д1	Система управління збору	1		
			даних у хмарному			
			середовищі			
			Схема структурна.			
	А1	ІАЛЦ.467200.006 Д2	Система управління збору	1		
			даних у хмарному			
			середовищі			
			Схема алгоритму.			
	А1	ІАЛЦ.467200.007 Д3	Система управління збору	1		
			даних у хмарному			
			середовищі			
			Діаграма класів.			

Змін	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467200.001 ОА		
Розроб.		Величко А.А.			Система управління збору даних у хмарному середовищі Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Сімоненко В.П.					
Н. контр.		Сімоненко В.П.				КПІ ім. Ігоря Сікорського ФІОТ, ІП-62	
Затвер.		Стіренко С.Г.					

№ з/п	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість	№	Примітки
	А4		Завдання на дипломний	3		
			проект.			
		ІАЛЦ.467200.002 ТЗ	Технічне завдання.	3		
	А4	ІАЛЦ.467200.004 ПЗ	Система управління збору	51		
			даних у хмарному			
			середовищі			
			Пояснювальна записка			
	А1	ІАЛЦ.467200.005 Д1	Система управління збору	1		
			даних у хмарному			
			середовищі			
			Схема структурна.			
	А1	ІАЛЦ.467200.006 Д2	Система управління збору	1		
			даних у хмарному			
			середовищі			
			Схема алгоритму.			
	А1	ІАЛЦ.467200.007 Д3	Система управління збору	1		
			даних у хмарному			
			середовищі			
			Діаграма класів.			

Змін	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467200.002 ВП		
Розроб.		Величко А.А.			Система управління збору даних у хмарному середовищі Відомість проекту	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Сімоненко В.П.					
Н. контр.		Сімоненко В.П.				КПІ ім. Ігоря Сікорського ФІОТ, ІП-62	
Затвер.		Стіренко С.Г.					

Технічне завдання до дипломної роботи

Зміст

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку системи управління збору даних у хмарному серидовищі. Область застосування: організація збору даних в великих кількостях на постійній основі.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є полегшення та покращення процесів організації збору даних в хмарному середовищі та моніторинг даного процесу.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з теорії і практики теорії розкладів, багатокритеріального планування.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробляємого продукту

- Можливість гунчкого налаштування параметрів зборів даних.
- Автоматичний розподіл ресурсів.
- Можливість горизонтального масштабування.
- Можливість підтримки високих навантажень

5.2. Вимоги до програмного забезпечення

- Операційна система Linux.

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення літератури	25.03.2020
Складання і узгодження технічного завдання	15.04.2020
Реалізація та оцінка алгоритмів збору даних	5.05.2020
Тестування зборів даних	10.05.2020
Допрацювання і налагодження програми	20.05.2020
Оформлення документації дипломної роботи	25.05.2020

Пояснювальна записка
до дипломного проекту
на тему: «Система управління збору даних у хмарному середовищі»

Київ – 2020 року

ЗМІСТ

ВСТУП

В сучасному світі для успішного ведення бізнесу, проведення наукових робіт, аналізу ефективності процесів та сотні інших потреб необхідно безліч різних даних та інформації. І найпростішим та найшвидшим способом отримати необхідний об'єм даних - це сканувати їх з веб ресурсів. Існує багато компаній, які спеціалізуються на зборі даних, що показує актуальність даної проблеми на даний момент.

Є безліч інформації та програмних комплексів, які допомагають та розповідають як зібрати інформацію з веб-сторінки. Однак вони спеціалізуються на тому, як обробити одну сторінку. І немає достатньо інформації і тим паче програмних комплексів, які вирішують проблему управління та планування збору мільйонів веб-сторінок з різних веб-ресурсів на постійній основі, орієнтуючись на те, коли та за скільки необхідно зібрати інформацію та на те, які навантаження витримує веб-ресурс. Особливою проблемою також постає те, що є обмежені ресурси, якими збираються дані.

Метою дипломної роботи є створення системи управління збору даних, яка орієнтуючись на необхідну кількість даних для збору та ефективності роботи веб-ресурсу правильно організувала швидкість збору та виділяла необхідну кількість ресурсів для їх збору.

Визначено, що для того щоб досягти вищезазначеної мети необхідно послідовно здійснити наступні пункти:

1. Проаналізувати предметну область, яка повністю охоплює зазначену тему.
2. Формалізувати постановку задачі, яку необхідно вирішити в результаті даної роботи.

3. Реалізувати пошук систем, які б вирішували поставлену в попередньому пункті проблему.
4. За наявності систем, які покривають дану задачу, проаналізувати їх роботу та зробити висновки про недоліки існуючих способів вирішення поставленої задачі.
5. Здійснити аналіз технологій, за допомогою яких може бути реалізована подібна система.
6. В результаті аналізу плюсів та мінусів наявних технологій, обрати найбільш доцільні технології для реалізації потрібної системи.
7. Створити систему управління збору даних у хмарному середовищі.
8. Провести тестування додатку, зібравши дані з визначених веб-ресурсів за певний проміжок часу

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАДАЧІ

1.1 Загальні відомості про збір даних з веб-ресурсів

Збір даних у своїй найбільш загальній формі - це техніка, в якій комп'ютерна програма витягує дані з результатів, згенерованих з іншої програми. Однак збір даних зазвичай визначають в зборі веб-сторінки, та процесі отримання цінної інформації з веб-сайту.

Зазвичай компанії не хочуть, щоб їх унікальний контент завантажувати та використовувати для несанкціонованих цілей. Як результат, вони не відкривають усі дані через API або інший легкодоступний ресурс. З іншого боку, веб-скрепери(програми, які займаються збором даних з веб-ресурсів) зацікавлені отримати дані веб-сайтів незалежно від будь-яких спроб обмеження доступу. В результаті чого існує гра - «коти-миші» між веб-скреперами, які збирають дані, та різними стратегіями захисту даних, кожен з яких намагається перевершити інший[6].

Процес збору веб-сторінок досить простий, хоча реалізація може бути і достатньо складною. Веб-скрапінг відбувається в 3 етапи:

1. Спочатку фрагмент коду, який використовується для витягування інформації надсилає HTTP GET-запит на необхідний веб-сайт.
2. Коли веб-сайт відповідає, веб-скрепер розбирає HTML-документ для конкретного шаблону даних.
3. Після вилучення даних, вони перетворюються у будь-який конкретний формат, який створив автор веб-скрепера.

   Найпопулярніші формати вилучених даних: JSON, CSV, XML.

   Веб-скрапери можуть бути розроблені для багатьох цілей, таких як:

1. Скрапінг вмісту - вміст можна витягнути з веб-сайту для того, щоб перетворити унікальну перевагу певного товару чи послуги, що спирається на вміст. Наприклад, такий продукт, як Yelp, покладається на відгуки, учасник може зібрати весь вміст відгуку з Yelp та відтворити вміст на власному веб-сайті, роблячи вигляд, що вміст є оригінальним.
2. Збір цін - шляхом збору даних про ціни, конкуренти мають змогу агрегувати інформацію про свою конкуренцію. Це дозволяє їм сформулювати унікальну перевагу.
3. Збір контактів - багато веб-сайтів містять адреси електронної пошти та номери телефонів у відкритому тексті. Скануючі такі місця, як онлайн-список працівників, скрепер може зібрати контактні дані для масових списків розсилки[11].

Зазвичай весь вміст, який відвідувач веб-ресурсу може побачити, повинен бути перенесений на машину відвідувача, тому будь-яку інформацію, до якої відвідувач може отримати доступ, можна зібрати за допомогою веб-скрапера.

Однак можна докласти зусиль, щоб обмежити кількість даних, які отримує програма. Ось 3 найпоширеніші способи обмеження впливу зусиль зі збору даних:

1. Обмеження кількості запитів за певний час - для відвідувача, який переглядає серію веб-сторінок на веб-сайті, швидкість взаємодії з веб-сайтом досить передбачувана. Наприклад, ви ніколи не будете переглядати 100 веб-сторінок в секунду. З іншого боку, комп’ютери можуть робити набагато більше запитів, ніж людина, а початківці в зборі даних можуть використовувати методи скрапінгу без обмежень на

швидкість, щоб спробувати зібрати весь веб-сайт дуже швидко. Обмежуючи швидкість максимальної кількості запитів, яку може задати певна IP-адреса протягом певного часу, веб-сайти можуть захистити себе від надмірної кількості запитів на сайт.

2. Зміна розмітки HTML через регулярні проміжки часу - веб-скрапери покладаються на послідовне форматування, щоб ефективно проаналізувати вміст веб-сайту та зберегти лише корисні дані. Одним із способів переривання цього робочого процесу є регулярна зміна елементів розмітки HTML, щоб послідовне зчитування ускладнювалося. Вкладаючи HTML-елементи або змінюючи інші аспекти розмітки, прості зусилля зі скорочення даних будуть затруднені або зовсім зірвані[12]. Для деяких веб-сайтів кожного разу, коли веб-сторінка надається, деякі форми захисту вмісту додають деякі випадкові частинки, які не впливають на відображення веб-сторінки для людини, однак ускладнюють процес збору даних для веб-скрапера. Інші веб-сайти час від часу навіть повністю змінюють код розмітки, щоб уникнути довготривалих спроб зі збору даних.
3. Використання CAPTCHA у випадку великої кількості запитів - окрім використання рішення, що обмежує швидкість, ще одним корисним кроком для уповільнення веб-скреперів є вимога, щоб відвідувач веб-сайту відповідав на складне для комп’ютера, але просте для людини запитання. Наприклад знайти велосипед на фотографії або розібрати текст, який написаний незрозумілим для комп'ютера шрифтом. Однак постійні виклики CAPTCHA можуть негативно впливати на користувацький досвід[8].

1.2 Огляд існуючих систем управління збору даних

Існує велика кількість компаній, які надають змогу збирати дані. Однак усі вони дають змогу безкоштовно збирати дані близько 1000 сторінок за місяць, а при необхідності збирати мільйони посилань кожний день мають плани, які надзвичайно дорогі. І до того ж вони не надають змогу керувати процесом збору даних.

Крім компаній, які самі займаються збором даних є велика кількість програм та фреймворків, які допомагають в процесі збору даних. Однак функціонал пов'язаний з організацією та планування достить скупий і зовсім не влаштовує нашим потребам.

Тож систем управління збору даних у вільному доступі зовсім немає або ж це частина продукту різних компаній, що спеціалізуються на зборі даних.

1.3 Проблеми, що виникають при створенні системи управління збору даних

При вирішенні даної задачі виникають проблеми, що пов'язані зі створенням швидкого та високоефективного процесу управління збору даних.

Однією з проблем створення вдалої та ефективної системи управління збору даних є оптимізація внутрішніх процесів на організацію збереження та передачі даних. Великою проблемою постають всі недоліки в передачі даних за допомогою мережі інтернет. А також правильний розрахунок необхідних ресурсів для збору даних. Для розуміння процесу збору даних також необхідно мати достатньо гарну систему моніторингу основних процесів, які відповідають за злагоджений процес збору[10].

Отже, можна підсумувати наступні проблеми, які вирішуються програмним комплексом:

- Відсутність зручного способу організувати збір великої кількості даних на веб-ресурсах.
- Вирішення непередбачуваних проблем пов'язаних з передачею даних за допомогою мережі Інтернет.
- Розрахунок необхідних ресурсів для безпечного збору даних.
- Повноцінний моніторинг всього процесу збору даних.

1.4 Постановка задачі

Метою виконання даного дипломного проекту є створення повністю автоматизованої системи управління збору даних в хмарному середовищі, яка полегшує та покращує процес збору даних.

Розуміючи недоліки та недостатність доступних рішень, можна сформувати основні вимоги до даної системи, а саме:

- Продуктивне розподілення ресурсів необхідних для збору даних
- Правильне планування порядку та швидкості збору, спираючись на зазначений час збору, обмеження з боку веб-ресурсів та мережі інтернет
- Ефективне збереження результатів збору даних
- Всеосяжний моніторинг процесів системи

Серед додаткових вимог, що можуть бути реалізовані після масштабування системи для широкого кола застосування можна зазначити:

- Передбачення можливостей конкретних веб-ресурсів у питанні збору даних спираючись на історичні дані попередніх зборів
- Розробка адміністративної панелі для більш детального налаштування зборів
- Запуск зборів в тих регіонах, де знаходяться дані, щоб мінімізувати витрати на передачу даних через мережу

ВИСНОВКИ ДО РОЗДІЛУ 1

Суть першого розділу полягала у зборі та аналізі наявної інформації про предметну область пов'язану зі збором даних у хмарному середовищі та проведенні аналізу вже існуючих рішень визначених проблем. Було здійснено огляд продуктів, які можуть задовольняти усім необхідним вимогам, однак їх функіонал покриває досить малу частину проблем, які потрібно вирішити для досконалого та швидкого збору даних. Проаналізувавши більшість проблем, які виникають в процесі збору даних, було вирішено також реалізувати повноцінний моніторинг компонентів системи збору, для подальшого аналізу етапів та проблем під час збору. Результатом цього розділу є сформована постановка задачі з детальним списком вимог до отриманого програмного продукту.

РОЗДІЛ 2
АНАЛІЗ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Засоби представлення даних

Комунікація сервісу з користувачем та самими веб-скареперами відбувається за допомогою текстового протоколу. Надзвичайно зручним в структуруванні даних та їх серіалізації та десеріалізації є формат JSON. Хоч формат даних, який передається даній системі є достатньо простою, формат JSON надає можливість в подальшому використовувати більш складну та розширену структуру повідомлення. Основні переваги даного формату:

- легкий у використанні.
- великий спектр підтримки форматів даних для передачів.
- уже є надзвичайно швидкі алгоритми обробки даних в даному форматі

Недоліки:

- відсутність повідомлень про помилку;
- велика вразливість даних.

Наступний приклад ілюструє базовий формат повідомлення, який приходить в систему для подальшого збору:

```
{
        "spider": "base:google.com",
        "tracking_id": "google-tracking",
        "url": "https://google.com",
        "scan_time": "2020-05-01T12:00:00",
        "check_point_time": "2020-05-01T15:00:00"
 }
```

В даному проекті використовується мова програмування Python, яка має вбудовану бібліотеку для роботи за даними в форматі JSON, тому за

допомогою виклику лише однієї функції, повідомлення в форматі JSON швидко та зручно перетворюється в об'єкт мови програмування Python з яким можно з легкістью продовжити роботу.

2.2 Засоби передачі даних

Для передачі вхідних та вихідних даних, а саме інформації по тому яку саме інформацію необхідно зібрати та результати зборів, необхідно використовувати систему для передачі повідомлень. Система, яка найбільше підходить для швидкої передачі великої кількості повідомлень є Apache Kafka. Kafka добре працює як традиційний брокер повідомлень. На відміну від більшості систем обміну повідомленнями, Kafka має кращу пропускну здатність, вбудований розділ, реплікацію та відмовостійкість, що робить її хорошим рішенням для широкомасштабних програм обробки повідомлень. Тобто використання Kafka надає більшу впевненість в цілісності та швидкій обробці повідомлень.

В той же час вона має дуже простий інтерфейс взаємодії. На малюнку(рис. 2.1) відображено осноні компоненти для взаємодії, а саме Producer(відповідає за відправлення даних) та Consumer(відповідає за читання даних). Kafka використовує Topic як окрму чергу для відповідного виду даних, а Partition необхідний для розподілу даних в одному топіку на деклькох машинах в кластері[3].

Kafka зберігає всі відправлені повідомлення та надає інтерфейс взаємодії, завдяки якому одне повідомлення може бути прочитане та опрацьоване різними процесами, що надає можливість використовувати повідомлення не тільки для передачі, а й для аналізу результатів зборів.

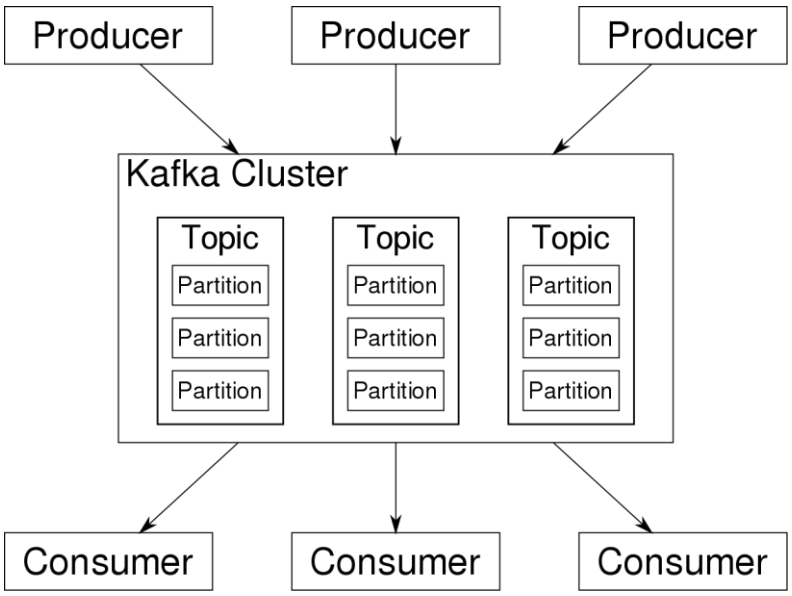


Рисунок 2.1 Алгоритм роботи Kafka

2.3 Засоби збереження даних

Для збереження вхідних(які веб-сторінки необхідно зібрати) та вихідних(результати зборів) даних необхідно мати швидку для читання та запису і відмовостійку базу даних. Адже збір даних досить затратна операція і ми не можемо дозволити втратити зібрані дані та уповільнювати процес збору на комунікацію з базою даних. Тому в якості такого рішення було обрано базу даних - Apache Cassandra. Cassandra - це масивно масштабована база даних NoSQL з відкритим кодом. Кассандра ідеально підходить для управління великою кількістю структурованих, напівструктурованих та неструктурованих даних у кількох центрах обробки даних та у хмарному середовищі. Cassandra забезпечує постійну доступність, лінійну масштабованість та операційну простоту на багатьох серверах без жодної точки відмови, а також потужна динамічна модель даних, розроблена для максимальної гнучкості та швидкого часу відгуку[4].

Cassandra побудована для масштабної архітектури, що означає, що вона здатна обробляти петабайти інформації та тисячі одночасних операцій в секунду. Це можливо завдяки таким основним підходам в збереженні даних:

1. Cassandra - це база даних із розділеними рядами. Рядки впорядковані в таблиці з необхідним первинним ключем. Зберігання рядків означає, що, як і реляційні бази даних, Cassandra впорядковує дані за рядками та стовпцями.
2. Автоматичний розподіл даних. Cassandra забезпечує автоматичний розподіл даних по всіх вузлах, які беруть участь у кластері бази даних. Немає нічого програмного, що розробнику чи адміністратору потрібно зробити або програмувати для розподілу даних по кластеру, оскільки дані прозоро розподілені по всіх вузлах кластера.
3. Вбудована та настроювана реплікація. Cassandra також забезпечує вбудовану реплікацію, яка зберігає зайві копії даних у вузлах, які беруть участь у кластері Cassandra. Це означає, що якщо будь-який вузол кластера виходить з ладу, то одна чи більше копій даних цього вузла доступні на інших машинах кластера. Реплікація може бути налаштована для роботи через один центр обробки даних, безліч центрів обробки даних та декілька областей доступності в хмарному середовищі.
4. Cassandra забезпечує лінійну масштабованість. Це означає, що продуктивність можна легко підвищити просто додаванням нових вузлів під час роботи кластеру. Наприклад, якщо 2 вузли можуть обробляти 100000 транзакцій в секунду, 4 вузли підтримують 200000 транзакцій в секунду, а 8 вузлів вирішать 400000 транзакцій в секунду.

   Тобто Cassandra повністю задовольняє потребі в заберіганні та читанні великої кількості даних та їх майже 100% збереження.

Однак також необхідно обмінюватись даними із самими веб-скраперами. І ці дані необхідні на короткотривалий проміжок часу і повинні бути настільки швидко доступні, ніби вони лежать в оперативній пам'яті, а не на диску. Для вирішення даної проблеми ідеально підійде Redis. Адже Redis - це система управління базами даних, яка зберігає дані в пам'яті. Він використовується як база даних, кеш-пам'ять та брокер повідомлень. Він підтримує структури даних, такі як рядки, хеші, списки, набори, відсортовані набори з діапазонами запитів, растрових зображень, геопросторових індексів з радіусними запитами та потоками[5]. Redis має вбудовану реплікацію, Lua скриптинг, LRU(Least recently used) витіснення, транзакції та різні рівні стійкості на диску, а також забезпечує високу доступність. З усього величезного списку можливостей Redis ми будемо використовувати збереження даних в структурі - множина, для якої є дуже зручний та швидкий інтерфейс для додавання та вилучення даних[13].

І для Cassandra, і для Redis існують бібліотеки для комунікації на різних мовах програмування, зокрема і для Python. За допомогою широкого функціоналу даних бібліотек, ми отримуємо простий та зручний інтерфейс взаємодії з цими базами даних та можем з легкістю використати всі переваги, які вони надають.

2.4 Організація моніторингу системи

Основоположними частинами побудови моніторингу є збереження та відображення даних. Для зберігання метрик які змінюються в часі необхідно використовувати базу даних часових рядів. Для цього іделаьно підійде база данних Prometheus. Вона збирає показники з налаштованих IP-адрес через задані інтервали, виконвати різні вирази, відображати їх результати та в

результаті може викликати сповіщення, якщо деяка умова не дотримується[9].

На малюнку(рис. 2.2) зображено схему використання Prometheus для збору метрик.

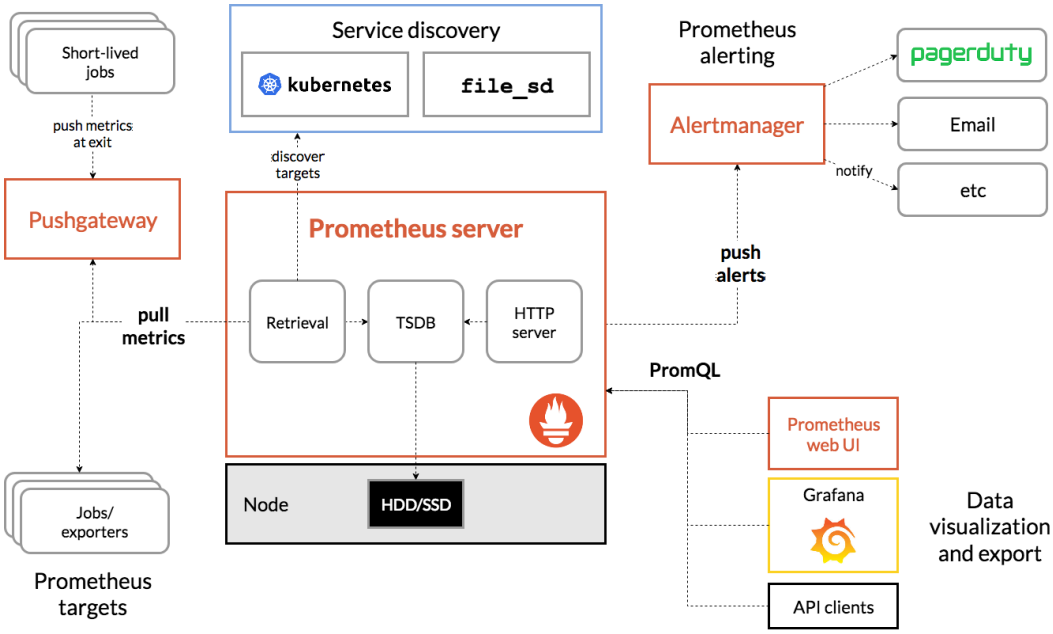


Рисунок 2.2 Процес взаємодії з Prometheus

Для того щоб відобразити зібрані метрики використовується система для відобрадення даних Grafana. Grafana - це рішення з відкритим кодом для збору аналітики, підтягуючи показники, для величезної кількості даних та допомагає моніторити програми за допомогою широко конфігурабельних інформаційних панелей.

Grafana з'єднується з усіма можливими джерелами даних, такими як Graphite, Prometheus, Influx DB, ElasticSearch, MySQL, PostgreSQL тощо. Grafana дозволяє також писати плагіни з нуля для інтеграції з декількома різними джерелами даних[15]. Інструмент допомагає нам вивчати, аналізувати та контролювати дані протягом певного періоду часу.

2.5 Засоби створення бізнес логіки

Бізнес-логіка – в розробці інформаційних систем – сукупність правил, принципів, залежностей поведінки об'єктів предметної області (сфери людської діяльності, яку підтримує система). Інакше можна сказати, що бізнес-логіка – це реалізація правил і обмежень поведінки системи. Є синонімом терміна «логіка предметної області».

Простіше кажучи, бізнес-логіка – це реалізація предметної області в інформаційній системі. До неї відносяться, наприклад, формули розрахунку щомісячних виплат по позиках (у фінансовій індустрії), автоматизоване надсилання електронного листа керівнику проекту після закінчення виконання частин завдання всіма підлеглими (в системах управління проектами), відмова від готелю при скасуванні рейсу авіакомпанією (в туристичному бізнесі) і т.д.[1]

Для повноцінної реалізації бізнес логіки, яка не буде залежати від інфраструктурних залежностей необхідно використовувати найкращі практики в організації архітектури програмного засобу. Прикладом такої архітектури може бути “Чиста Архітектура”.

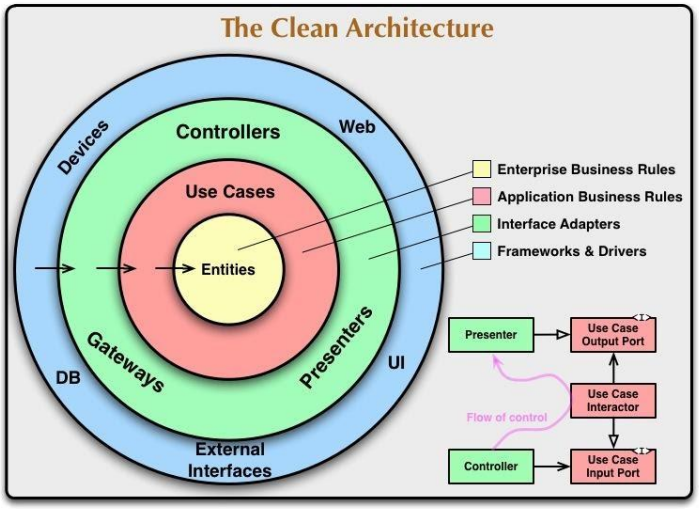


Рисунок 2.3 Чиста архітектура

Основними характеристиками чистої архітектури є:

- Незалежні рамки. Архітектура не залежить від існування певної бібліотеки програмного забезпечення, що навантажується ознаками. Це дозволяє використовувати такі фреймворки як інструменти, замість того, щоб втиснути систему в обмеження.
- Тестування. Бізнес-правила можуть бути перевірені без інтерфейсу користувача, бази даних, веб-сервера або будь-якого іншого зовнішнього елемента.
- Незалежність від UI. Інтерфейс користувача може легко змінюватися, не змінюючи решту системи. Веб-інтерфейс користувача може бути замінений інтерфейсом консолі, наприклад, без зміни бізнес-правил.[2]
- Незалежна база даних. Існує можливість змінити Oracle або SQL Server, для Mongo, BigTable, CouchDB або щось інше. Бізнес-правила не пов'язані з базою даних.
- Незалежність від будь-якого зовнішнього агентства. Насправді бізнес-правила просто нічого не знають про зовнішній світ.

ВИСНОВОКИ ДО РОЗДІЛУ 2

Основне завдання другого розділу полягає у проведенні аналізу використаних в процесі розробки технологій. Прийняття рішення про використання підходящих способів обробки, передачі та збереження даних, що є ключовими моментами в організації системи по управлінню збору даних. Таким чином, його результатом став аналіз переваг та недоліків вибраного програмного забезпечення та підходів у побудові архітектури застосунку. І даний аналіз показав, що усі застосовані технології та підходи є доцільними для досягнення мети дипломної роботи.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ УПРАВЛІННЯ ЗБОРУ ДАНИХ

3.1. Загальний огляд проекту

Для повноцінного виконання проекту з використанням кращих практик з побудови архітектури, його було розділено на основні компоненти:

1. Зчитування та обробка вхідних даних
2. Комунікація з веб-скраперами
3. Розрахунок необхідних ресурсів та планування збору даних
4. Відправка даних для моніторингу
5. Підготовка та відправлення результатів збору
6. Сервер

Для дотримання підходу Чистої Архітектури, взаємодія з базами даних винесена окремо і вона не впливає на реалізацію бізнес логіки.

Архітектура проекту в середовищі розробки PyCharm Professional наведена на рис. 3.1

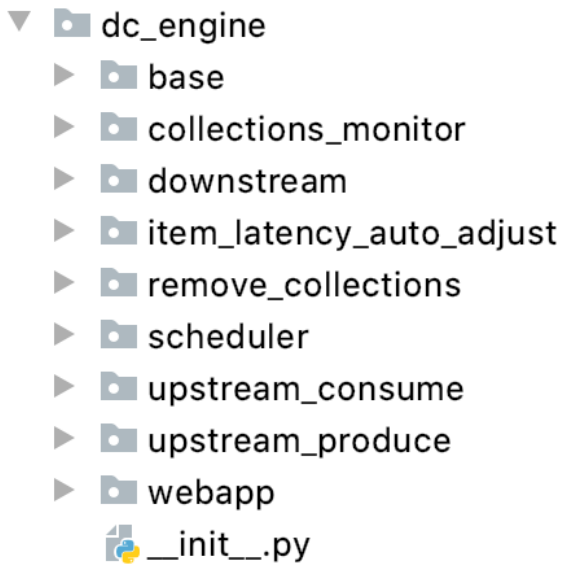


Рисунок 3.1 Архітектура проекту в середовищі розробки

3.2 Огляд основних компонентів системи

Загальна структуру взаємодії всіх компнонтів системи наведено в Додатку А.

3.2.1 Обробка вхідних даних

Процес збору даних починається з того, що необхідно отримати інформацію про те, що саме нам потрібно збирати. Ці дані необхідно провалідувати та підготувати всі необхідні залежності для ініціалізації збору необхідних даних. Дані, які характеризують, що саме необхідно зібрати всередині системи мають назву - трекінг. Перш за все потрібно створити колекцію, яка буде відповідати збору набору даних по відповідному веб-ресурсу за вказаний час. І далі додати отриману інформацію для збору в список трекінгів для збору.

Отже, проаналізувавши всі етапи обробки вхідних даних було створено клас UpstreamConsumeIntreface, який має наступний інтерфейс:

1. run - метод який створює дві співпрограми, одна з який зчитує повідомлення з Kafka, а інша записує результати в Cassandra. А також відповідає за їх коректне завершення в разі зупинки виконання.
2. consume_msgs - метод, який зчитує повідомлення з Kafka
3. insert_msgs - метод, що відповідає за запис трекінгів в Cassandra

Тобто процес обробки вхідних повідомлень представляє собою зчитування повідомлень з Kafka, валідацію цього повідомлення за допомогою трафарету та подальшу конвертацію даних в об'єкт трекінгу зі статусом NEW і збереження його у внутрішній черзі. Формат валідації повідомлення має вигляд:

```
{
   "spider": trafaret.String(min_length=1),
   "tracking_id": trafaret.String(),
```

```
        "url": trafaret.URL,
        "meta": trafaret.Any,
        "scan_time": UtcDatetimeTrafaret(),
        "check_point_time": UtcDatetimeTrafaret(),
    }
```

Далі об'єкти трекінгу з цієї черги беруться для запису в базу даних, при чому правильно їх групуючи для виконання Batch Statement в Cassandra, що потребує дані які будуть додані тільки по одному ключу розділення.

3.2.2 Комунікація з веб-скраперами

Після того як ми отримали інформацію по тому, які саме дані нам необхідно зібрати, нам необхідно відправити ці дані безпосередньо для веб-скраперів, які будуть збирати інформацію. А також забирати дані, які були отримані під час роботи веб-скрапера. Оскільки під час збору даних можуть виникати безліч непередбачуваних помилок, то також необхідно реалізувати функціонал перевідправлення інформації для збору у випадку коли ми довго не отримуємо результат збору.

Для реалізації даного функціоналу було створено компонент DownstreamInterface, який працює в 3 етапи:

1. Забрати зібрані дані із Redis, куди веб-скрапери зберігають дані. Далі ці дані зберігаються в Cassandra. Та статус трекінга переходить в SCRAPED. Однак, якщо ми не отримали коректні дані, то трекінг переходить в статус ERROR. І останній крок - це оновлення часу, коли ми зібрали дані по колекції.
2. Необхідно відправити дані для збору. А саме трекінги зі статусом NEW та ERROR формуються у відповідний формат для веб-скраперів та кладуться в Redis. А статус трекінга переходить в QUEUED. Якщо є трекінги зі статусом ERROR, але кількість спроб зібрати трекінг

перевищує за ліміт, то ми не відправляємо їх, а переводимо в статус DISMISSED_BY_RETRIES.

3. Відібрати всі колекції, по яким за певний проміжок часу(за замовчуванням 10 хвилин) не отримано жодних даних, що скоріш за все означає, що процес збору веб-ресурсу зазнав непередбачуваних проблем, і по таким колекціям відібрати всі трекінги, що знаходяться в статусі QUEUED та перевідправити їх в Redis для повторного збору.

В результаті роботи даного компоненту, ми отримуємо швидкий та простий спосіб комунікації із веб-скраперами, який до того ж реалізує функціонал перевідправлення даних.

3.2.3 Розрахунок необхідних ресурсів та планування збору даних

Основна складність всього процесу управління зборами є правильність розрахунку необхідних ресурсів для збору даних враховуючи час за який потрібно провести весь процес, обмеженість веб-ресурсу на якому збираються дані та кількість даних для збору. Тому було розроблено компонент Scheduler який покриває дані проблеми.

Перший етап - це підрахунок необхідних ресурсів. Основне число яке нам необхідно отрмати - це кількість одночасних запитів на веб-ресурс, які необхідно робити, щоб встигнути зібрати всі дані за певний час. Це число розраховується за формулою:

$$\mathrm{C} = \frac{(T * L)}{(t_e - t_s)},$$

де C – кількість одночасних запитів;

T – кількість трекінгів, які необхідно зібрати;

L – час збору одного трекінгу, с;

t_e – час закінчення збору, с;

t_s – час початку збору, с;

В результаті ми отримуємо число всіх одночасних запитів які повинна виконувати система. Однак це число обмежується максимальною кількістю запитів, які можна робити на веб-ресурс, адже якщо ми отримаємо занадто велике число, то даний веб-ресурс перестане функціонувати. Надалі отриману кількість запитів ми розподіляємо на всі доступні машини, з розрахунком на те, що 1 машина може робити, по замовчуванні, 100 одночасних запитів. Тому при необхідності робити 300 одночасних запитів, ми розподіляємо роботу одночасно на 3 машини, кожна з яких буде робити по 100 запитів. Даний перерахунок відбувається кожну хвилину, адже веб-ресурс може в будь який момент почати швидше або повільніше відповідати на наші запити.

Наступним кроком є отримання доступних для збору машин. В даному дипломному проекті використовуються хмарні ресурси представлені компанією Amazon - AWS(Amazon Web Services), однак завдяки виділенням комунікації в окремий шар, ми з легкістю можемо замінити використання AWS на аналогічні рішення від провайдерів хмарних ресурсів таких як Google Cloud, Microsoft Azure, Oracle Cloud та багато інших. Отримавши доступні для нас машини, ми розподіляємо підраховані раніше кількості запитів на всі машини і за допомогою SDK від Amazon запускаємо необхідну кількість машин.

В кінці ми зберігаємо те, як ми розподілили збори серед наявних ресурсів, для того щоб на наступному циклі роботи найбільш ефективно розподілити ресурси.

3.2.4 Підготовка та відправлення результатів збору

Після того як дані були зібрані веб-скраперами та збережені до бази даних, їх необхідно відправити клієнту. Дану роль виконує компонент UpstreamProduceInterface. Він збирає всі трекінги, що знаходиться в статусі SCRAPED, бере по ним зібрані дані та відправляє сформоване повідомлення в Kafka, звідки клієнт уже їх отримає. Якщо є трекінги в статусі DISMISSED_BY_RETRIES, то для них формується повідомлення, що на жаль не вийшло зібрати необхідні дані. Після відправки трекінги переходять в статус DELIVERED.

3.2.5 Відправка даних для моніторингу

Оскільки дана система орієнтована на те, щоб збирати одночасно дані з багатьох веб-ресурсів нам необхідно розуміти в якому статусі знаходиться кожен зі зборів. Для цього потрібен компонент який буде збирати статистику та відправляти її для подальшого відображення. Це питання вирішує компонент CollectionsMonitor, який збирає інформацію по тому скільки трекінгів знаходиться в кожному зі статусів NEW, QUEUED, SCRAPED, ERROR, DISMISSED_BY_RETRIES, DELIVERED і відправляє ці дані до Prometheus. Також кожен із попередніх сервісів відправляє свої дані для моніторингу, а саме UpstreamConsumeIntreface - кількість прочитаних повідомлень, UpstreamProduceInterface - кількість відправлених результатів зборів, Scheduler - число підрахованих паралельних запитів для кожної машини. Далі за допомогою функціоналу по будуванню графіків в Grafana ми відображаємо статус всіх зборів, а саме:

1. Скільки і в якому статусі знаходяться всі трекінги для конкретної колекції, де ми бачимо як з часом змінювався процес збору даних

Рисунок 3.2 Кількість трекінгів в кожному статусі

2. Загальну кількість трекінгів по всіх зборах разом

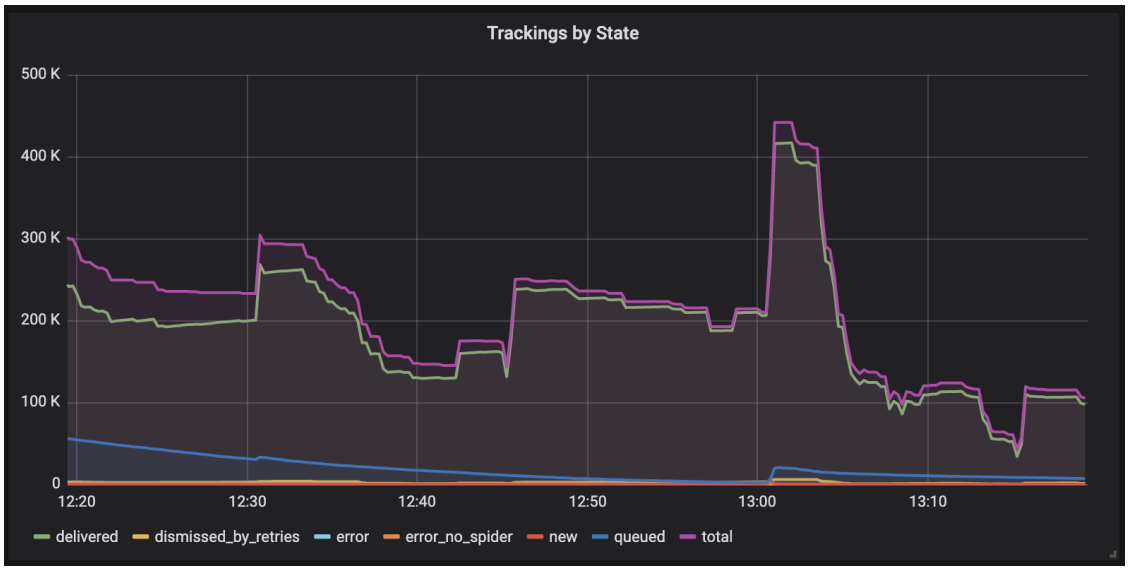


Рисунок 3.3 Загальна кількість трекінгів по всій системі

3. Показано, як змінювалося число паралельних одночасних запитів для збору колекції, та якій машині було надано конкретне число запитів на опрацювання

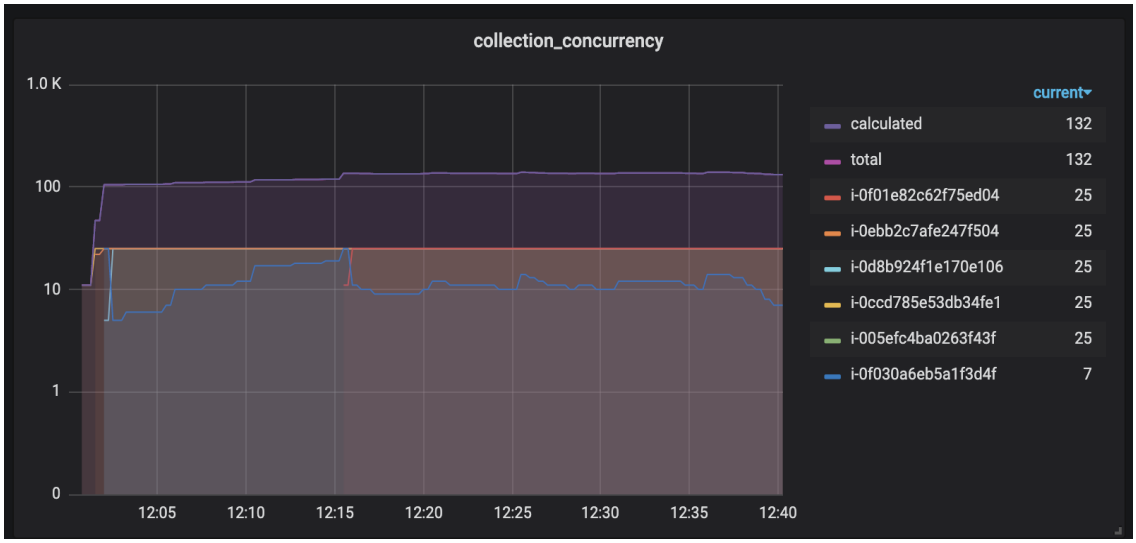


Рисунок 3.4 Розподіл навантаженя серед машин

4. Кількість прочитаних повідомлень від клієнта, та кількість відправлених йому відповідей

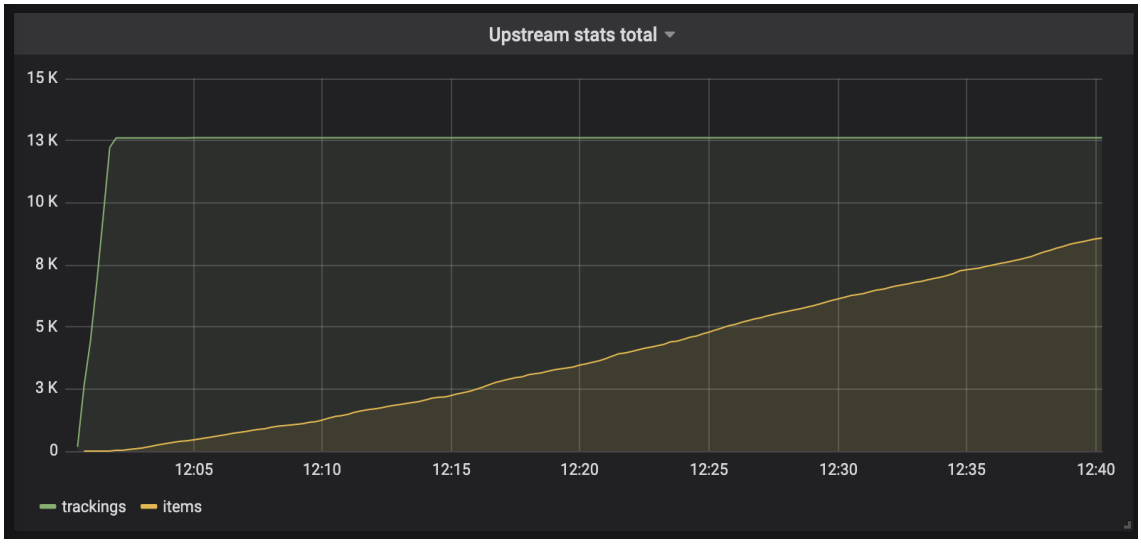


Рисунок 3.5 Прочитані та відправленні повідомлення

5. Яка кількість посилань знаходилась в Redis для опрацювання веб-скрапером, та скільки зібраних посилань уже знаходиться в Redis

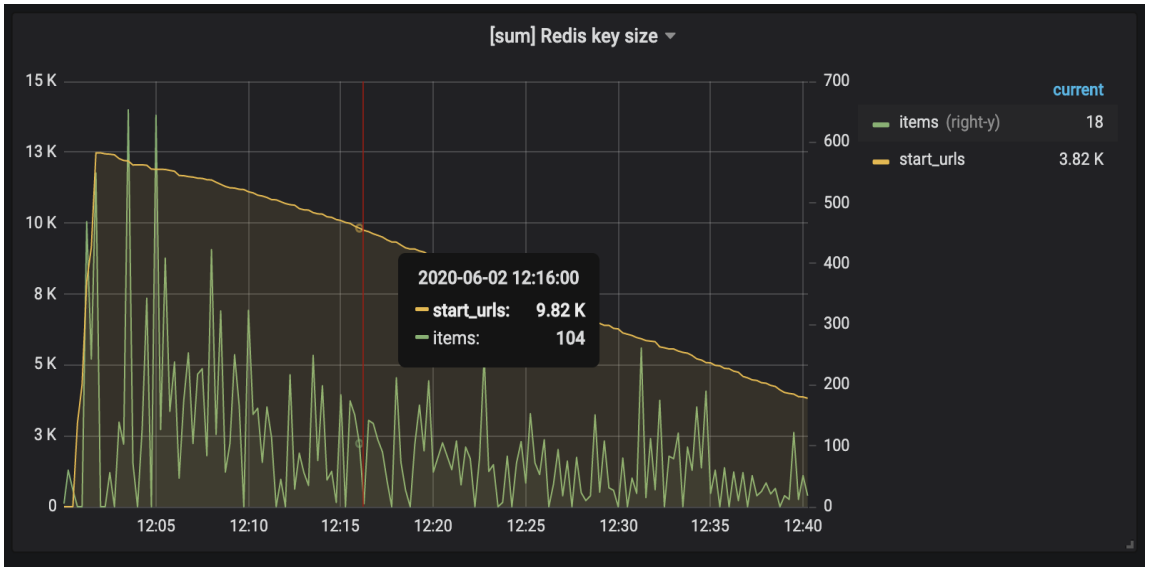


Рисунок 3.6 Кількість посилань для збору та зібрана інформація

3.2.6 HTTP Сервер

Для того, щоб мати можливість дізнатися про статус системи, а також внести налаштування для збору, реалізовано HTTP сервер, який має такі методи:

- GET /api/v1/spiders/ - дістати список доступних веб-скраперів
- POST /api/v1/spiders/ - додати новий тип веб-скрапера
- GET /api/v1/settings/ - отримати налаштування веб-скрапера
- POST /api/v1/settings/ - встановити налаштування веб-скрапера
- DELETE /api/v1/settings/ - видалити налаштування веб-скрапера
- GET /api/v1/crawler-settings/ - отримати інфорації про те, які веб-скрапери знаходяться на машині
- GET /api/v1/collection-status/ - отримати статистику по колекції

3.3 Специфікація бази даних

У наступних таблицях відображено специфікацію кожної одиниці моделі

Таблиця 3.1 Колекція

collection		
Поле	Опис	Тип
start	Час початок збору в ISO 8601 форматі	text
stop	Час кінця збору в ISO 8601 форматі	text
spider	Назва веб-скрапера	text
client_id	Ідентефікатор клієнта	text
id	Ідентефікатор колеції	text
PRIMARY KEY - (start, stop, spider, client_id)		

Таблиця 3.2 Додатокова інформація по колекції

collection_meta		
Поле	Опис	Тип
collection_id	Ідентефікатор колеції	text
li_ts	Час коли останній раз отримано дані від веб-кравлера	timestamp
lt_ts	Час коли останній раз отримано дані для збору від клієнта	timestamp
lr_ts	Час коли останній раз було виконано перевідправлення даних для збору	timestamp
PRIMARY KEY - (collection_id)		

Таблиця 3.3 Трекінг

tracking		
Поле	Опис	Тип
collection_id	Ідентефікатор колекції	text
shard	Шард	tinyint
state	Статус	tinyint
id	Ідентефікатор	text
meta	Додаткова інформація	text
spider	Ідентефікатор веб-кравлера	text
url	Посилання для збору	text
tires	Лічильник кількості спроб збору	tinyint
PRIMARY KEY - ((collection_id, shard), id)		

Таблиця 3.4 Одиниця збору

item		
Поле	Опис	Тип
collection_id	Ідентефікатор колекції	text
shard	Шард	tinyint
tracking_id	Ідентефікатор трекінгу	text
id	Ідентефікатор	text
spider	Ідентефікатор веб-кравлера	text
value	Інформація, що була зібрана з веб-ресурсу	text
PRIMARY KEY - ((collection_id, shard), tracking_id, id)		

Таблиця 3.5 Статус розположення веб-кравлерів

active_state		
Поле	Опис	Тип
id	Ідентифікатор	tinyint
ts	Час підрахунку	timestamp
value	Інформація про розположення веб-кравлерів	text
PRIMARY KEY - (id, ts)		
CLUSTERING ORDER - ts DESC		

Таблиця 3.6 Веб-кравлер

spiders		
Поле	Опис	Тип
name	Назва	text
cls	Ідентифіктаор	text
PRIMARY KEY - (name)		

Таблиця 3.7 Налаштування веб-кравлера

settings		
Поле	Опис	Тип
spider_name	Ідентифікатор веб-кравлера	text
name	Назва	text
value	Значення	text

Таблиця 3.7 (продовження)

is_runtime	Позначка, що вказує на необхідність перезавантаження веб-кравлера у випадку зміни	boolean
PRIMARY KEY - (spider_name, name)		

Таблиця 3.8 Налаштування веб-кравлера на конкретній машині

active_settings		
Поле	Опис	Тип
crawler	Ідентифікатор машини	text
spider_name	Ідентифікатор веб-кравлера	text
name	Назва налаштування	text
value	Значення налаштування	text
PRIMARY KEY - (crawler, spider_name, name)		

Таблиця 3.9 Статус колекції

collection_stats_ts		
Поле	Опис	Тип
id	Ідентифікатор колекції	text
ts	Час підрахування	timestamp
value	Інформація про	text
PRIMARY KEY - (id, ts)		
CLUSTERING ORDER - ts DESC		

3.4 Вимоги до технічного забезпечення

Для повноцінного функціонуванная системи управління збору даних нобхідно сервери баз даних, один або декілька серверів додатку та деклька серверів для веб-кравлерів.

3.4.1 Вимоги до серверу бази даних Cassandra

Як і більшість баз даних, пропускна здатність Cassandra покращується за допомогою більшої кількості ядер процесора, більшої кількості оперативної пам’яті та швидших дисків. Незважаючи на те, що Cassandra можна запускатися на невеликих серверах для тестувого або продакшн середовища розробки, для мінімального виробничого сервера потрібно щонайменше 2 ядра та принаймні 8 Гб оперативної пам’яті. Типові сервери виробництва мають 8 і більше ядер і щонайменше 32 ГБ оперативної пам’яті.

Cassandra вміє обробляти безліч одночасних запитів (і читання, і запису), використовуючи декілька потоків, що працюють на якомога більшій кількості процесорних ядер. Шлях запису даних на диск, як правило, сильно оптимізований, тому записи залежить до процесорного зв’язку. Отже, додавання додаткових процесорних ядер часто збільшує пропускну здатність як для читання, так і для запису.

Кассандра працює в Java VM, яка попередньо виділить heap фіксованого розміру (системний параметр xvax Java). На додаток до heap Cassandra використовуватиме значну кількість позапланової пам’яті оперативної пам’яті для стиснення метаданих, фільтрів Блума, кеш рядків, ключів та лічильників, а також кеш сторінок процесу. Нарешті, Cassandra скористається кешем сторінок операційної системи, зберігаючи недавно доступні файли порцій у оперативній пам’яті для швидкого повторного використання.

Для оптимальної роботи адміністратори повинні перевірити та налаштовувати свої кластери, виходячи з їх індивідуального навантаження. Однак основні вказівки пропонують:

- Оперативну пам’ять ECC завжди слід використовувати, оскільки Cassandra має кілька внутрішніх гарантій захисту від корупції на рівні бітів
- Heap Cassandra повинен становити не менше 2 ГБ і не більше 50% оперативної пам'яті вашої системи

3.4.2 Вимоги до серверу брокера повідомлень Kafka

Kafka багато в чому покладається на файлову систему для зберігання та кешування повідомлень. Всі дані негайно записуються в стійкий журнал файлової системи, не обов'язково передаючи на диск. Насправді це просто означає, що дані переноситься в кеш-пам'ять ядра. Сучасна ОС із задоволенням переадресує всю вільну пам'ять на кешування диска з невеликим погіршенням продуктивності, коли пам'ять буде відновлена. Крім того, Kafka дуже акуратно використовує heap і не вимагає встановлення розмірів heap більше 6 Гб[14].

Потрібна достатня пам’ять, щоб захистити активних читачів і записувачів. Машина з 64 ГБ оперативної пам’яті є гідним вибором, але машини на 32 ГБ - не рідкість. Менш ніж 32 Гб, як правило, є контрпродуктивним (знадобиться багато маленьких машин).

Більшість розгортань Kafka, як правило, досить легкі щодо потреб процесора. Точна настройка процесора має значення менше, ніж інші ресурси.

Вам слід вибрати сучасний процесор з декількома ядрами. Загалом кластери використовують 24 ядра на всіх машинах. Якщо вам потрібно вибрати більш швидкі процесори або більше ядер, виберіть більше ядер.

Додаткова конкурентоспроможність, яку пропонує кілька ядер, набагато перевищить трохи швидшу тактову частоту.

3.4.3 Вимоги до сервера додатка

Оскільки вся система розподілена на декілька незалежних компонентів, то для кожного з них можна виділити окремий сервер для більшої продуктивності. Однак в цьому немає необхідності та одного серверу з 4 ядерним процесором та 8 гігабайтами оперативної пам’яті буде достатньо для високопродуктивного функціонування всієї системи. Неймовірним важливим параметром є пропускна здатність мережі, поскільки проходиить дуже велика кількість операцій пов’язаних з передачею інформації, то бажано використовувати мережеву карту, яка має пропускну здатність більшу ніж 100 Мбіт/с.

3.4.4 Вимоги до серверів веб-кравлерів

Процес парсингу сайту не потребує багато ресурсів, тому для 1 веб-кравлера підійде машина з 1 ядрним процесором та 2 гігабайтами оперативної пам’яті. Однак кількість таких серверів бажано мати більше 20, поскільки система вмикає дані сревери тільки при необхідності, то збільшення кількості вже готових серверів сильно не впливває.

ВИСНОВОК ДО РОЗДІЛУ 3

Результатом даного розділу є готова специфікація по тому як працює даний програмний комплекс, та наведено підхід до його використання. Було досліджено, які ресурси, а сервери баз даних та на яких виконується сама програма, необхідні для повноцінної та безперебійної роботи. Внаслідок детального опису кожного компонента функціонування програми можна зробити висновок, що система збору даних справно працює та відповідає вимогам, що були сформовані. Виведено основні критерії моніторингу даної системи.

ЗАГАЛЬНІ ВИСНОВКИ

В ході виконання дипломного проекту було розглянуто питання пов’язані зі збором даних у хмарному середовищі, а також описано проблеми організації ефективного збору даних.

Після детальниого аналізу предметної області даної системи та її компонентів визначено постановку задачі та основні вимоги, яких необхідно дотриматись в процесі реалізації.

Розроблена система управління збору даних у хмарному середовищі, орієнтуючись на необхідну кількість даних для збору та ефективніс роботи веб-ресурсу, автоматично планує небхідну швидкість та кількість ресурсів для успішного збору даних. Це дозволяє з легкістю огрганізовувати одночасний збір мільйонів посилань на сотні веб-ресурсів.

Обширна система моніторингу дозволяє легко та ефективно прослідкувати за процесом збору та станоми системи загалом. Спираючись на зібрану статистку можна виявляти проблеми, з якими можна зіткнутися під час збору, ще до його початку та правильно налаштовувати весь процес.

Система управління збору була розроблена з використання найефективніших та відмовостійких технологій, що надає змогу витримувати надзвичайно великі навантаженн і з легкістю проводити горизонтальне масштабування, в разі необхідності.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Роберт С. Мартин. «Быстрая разработка программного обеспечения: принципы, примеры, практика». — М.: ООО «И.Д. Вильямс», 2004. — 752 с.
2. Роберт С. Мартин. «Чистый код: создание, анализ и рефакторинг.» — СПб.: Питер, 2010. — 464 с.
3. Нархид Н., Шапира Г., Палино Т. «Apache Kafka. Потоковая обработка и анализ данных» — СПб.: Питер, 2020. — 320 с.
4. Джефф Карпентер, Эбен Хьюитт «Cassandra. Полное руководство»: ДМК Пресс, 2017 — 400 с.
5. Джосіан Карлсон «Redis in Action». — Manning, 2013— 293 с.
6. Rayn Mitchel «Web Scraping with Python: Collecting Data from the Modern Web». — O'Reilly', 2015. — 256 с.
7. Eric Evans. Предметно-ориентированное проектирование (DDD): структура сложных программных систем — O'Reilly', 2011. — 448 с.
8. Richard Lawson «Web Scraping with Python: Successfully scrape data from any website with the power of Python». — O'Reilly', 2015. — 153 с.
9. Brian Brazil «Prometheus: Up & Running: Infrastructure and Application Performance Monitoring». — O'Reilly', 2018. — 356 с.
10. Gábor László Hajba «Website Scraping with Python: Using BeautifulSoup and Scrapy». — Apress, 2018 — 217 с.
11. Max Schallwig. «Web Scraping In Python: Master The Fundamentals». — Stone River eLearning, 2006. — 144 с.
12. Simon Munzert, Christian Rubba, Peter Meissner, Dominic Nyhuis. «Automated Data Collection with R: A Practical Guide to Web Scraping and Text Mining». — Wiley, 2015. — 449 с.

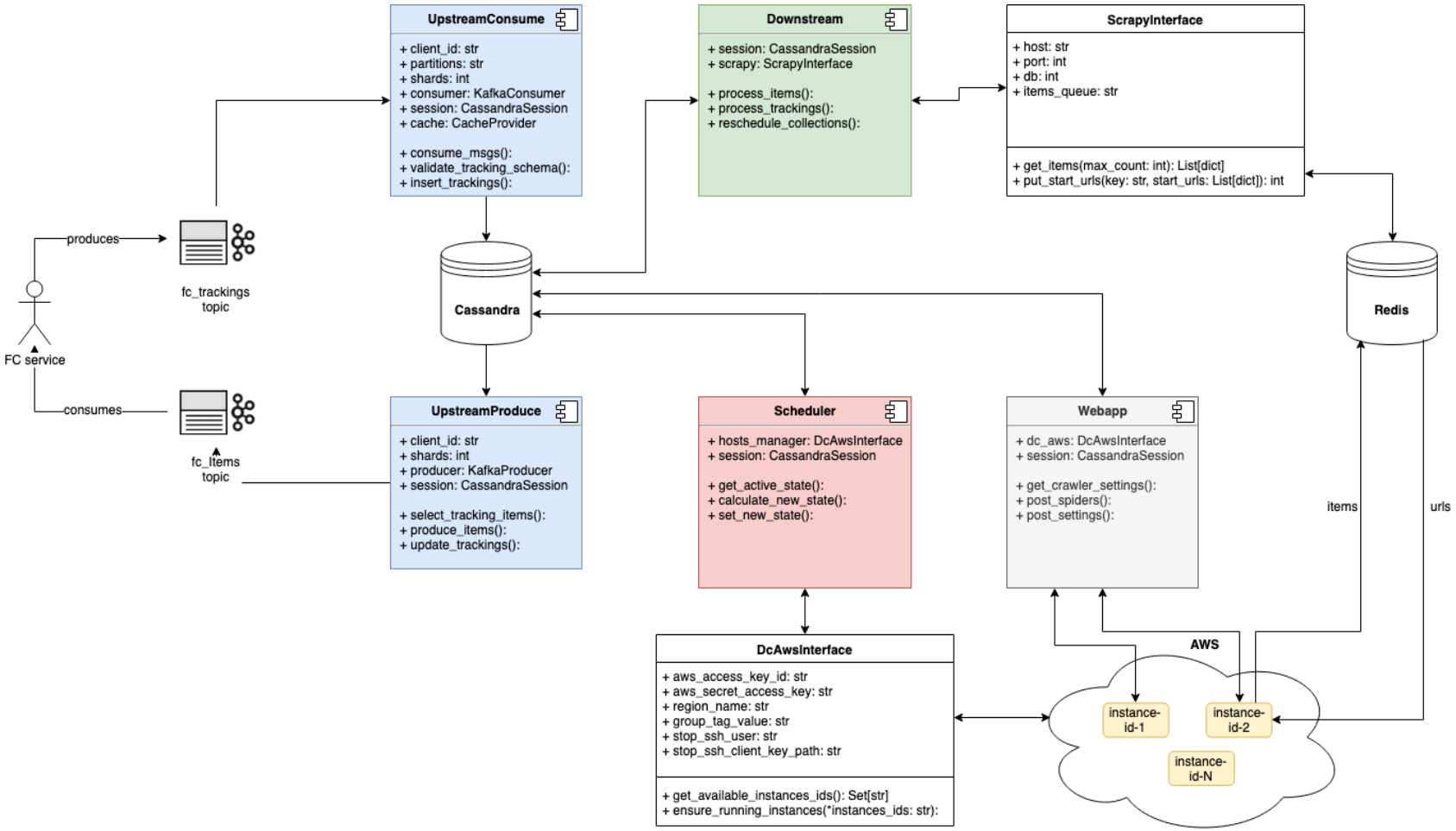
13. Redis documentation [Електронний ресурс] — Режим доступу: https://redis.io/documentation
14. Apache Kafka documentation [Електронний ресурс] — Режим доступу: https://kafka.apache.org/documentation/
15. Grafana documentation [Електронний ресурс] — Режим доступу: https://grafana.com/docs/grafana/latest/getting-started/what-is-grafana/

ДОДАТОК 1
Система управління збору даних у хмарному середовищі

Система управління збору даних у хмарному середовищі
Схема структурна
ІАЛЦ.467400.005 Д1

Аркушів 1

Київ 2020 р.



					ІАЛЦ.467200.005 Д1
Змін.	Арк.	№ докум.	Підпис	Дата	
Розроб.		Величко А.А.			Система управління збору даних у хмарному середовищі Струтурна схема
Перевір.		Валерій СІМОНЕНКО			
Н. контр.		Валерій СІМОНЕНКО			
Затвер.		Сергій СТІРЕНКО			

Літ.	Аркуш	Аркушів
	1	1

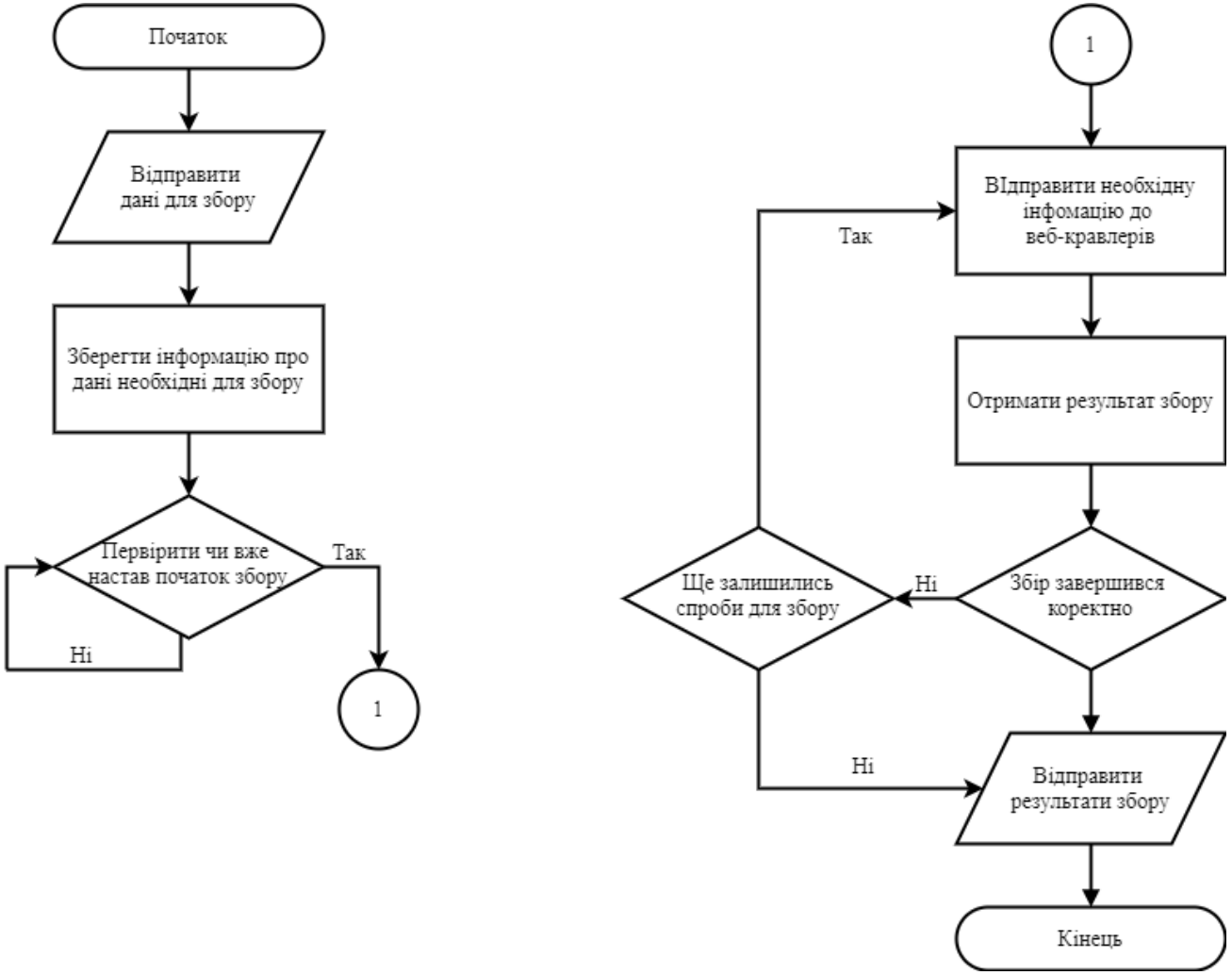
КПІ ім. Ігоря Сікорського
ФІОТ, ІП-62

ДОДАТОК 2
Система управління збору даних у хмарному середовищі

Система управління збору даних у хмарному середовищі
Схема алгоритму
ІАЛЦ.467400.006 Д2

Аркушів 1

Київ 2020 р.



					ІАЛЦ.467200.006 Д2
Змін.	Арк.	№ докум.	Підпи	Дата	
Розроб.		Величко А.А.			Система управління збору даних у хмарному середовищі Схема алгоритму
Перевір.		Валерій СІМОНЕНКО			
Н. контр.		Валерій СІМОНЕНКО			
Затвер.		Сергій СТІРЕНКО			

Літ.	Аркуш	Аркушів
	1	1

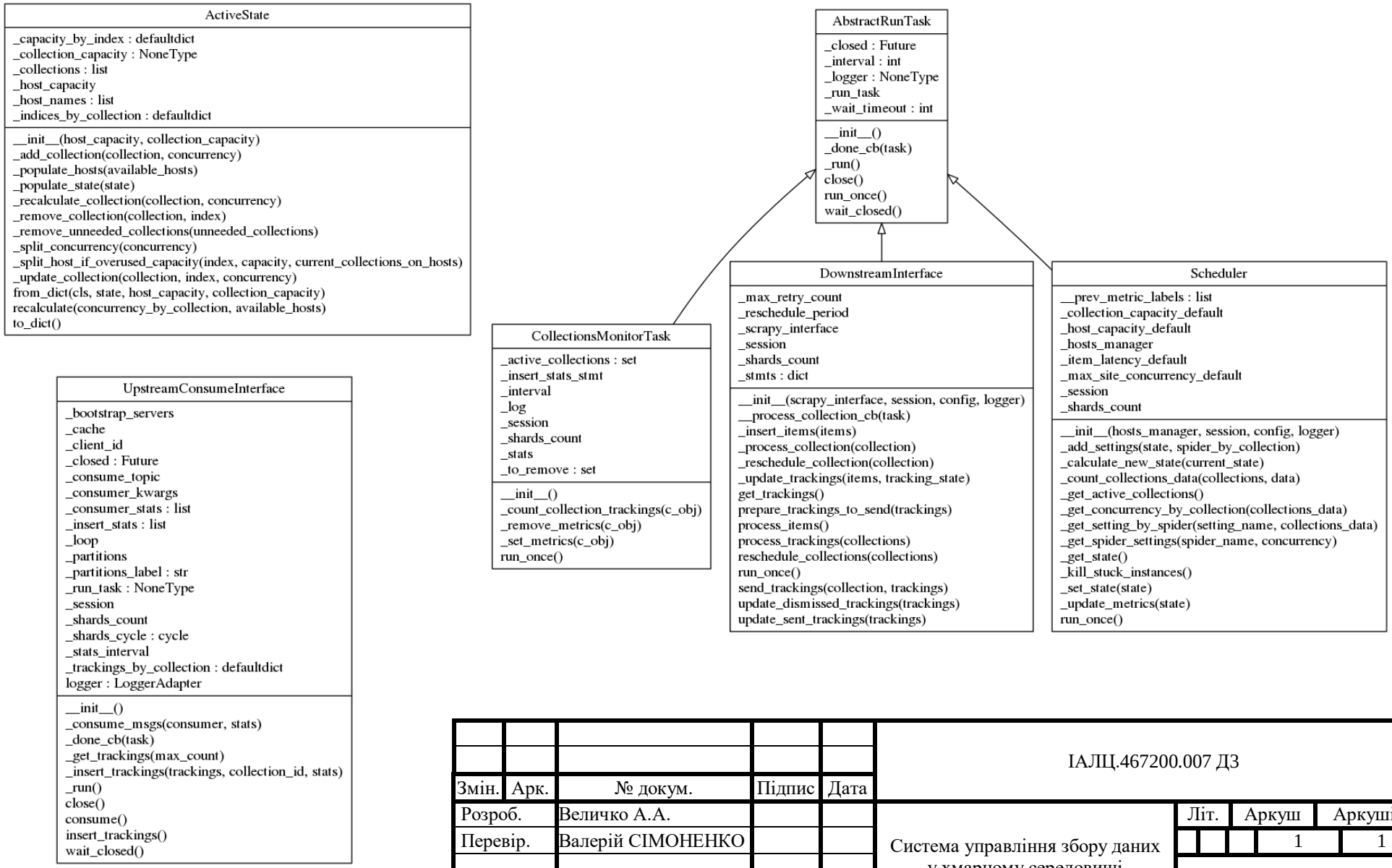
КПІ ім. Ігоря Сікорського
ФІОТ, ІП-62

ДОДАТОК З
Система управління збору даних у хмарному середовищі

Система управління збору даних у хмарному середовищі
Діаграма класів
ІАЛЦ.467400.007 Д3

Аркушів 1

Київ 2020 р.



					ІАЛЦ.467200.007 ДЗ	
Змін.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Величко А.А.			Система управління збору даних у хмарному середовищі Діаграма класів	Літ. / Аркуш 1 / Аркушів 1
Перевір.		Валерій СІМОНЕНКО				
						КПІ ім. Ігоря Сікорського ФІОТ, ІП-62
Н. контр.		Валерій СІМОНЕНКО				
Затвер.		Сергій СТІРЕНКО				